View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Elsevier - Publisher Connector

Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 62 (2015) 47 – 54

Procedia Computer Science

The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

Real-Time Languages, Timed Alternating Automata, and Timed Temporal Logics: Relationships and Specifications

Abdelaziz Fellah*

Dept. of Computer Science & Applied Statistics, University of New Brunswick, Saint John, New Brunswick, E2L 4L5, Canada

Abstract

Time dependent models have become increasingly important in formal modeling and verification of software systems. In this paper, we investigate the expressiveness of real-time iterative arrays and trees in the context of language recognition. Such synchronous parallel models of computation must meet high dependability requirements in performing complex interactions in a real-time environment. Furthermore, we present timed event alternating automata which do not only describe a theoretical model framework for formal languages but establishes strong relationships and foundations to timed propositional temporal logic through two embedded powerful metaphors – alternation and time.

© 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Peer-review under responsibility of organizing committee of The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)

Keywords: Real-time languages; timed alternating automata; timed metric temporal logic; iterative arrays and trees

1. Introduction and Related Work

Finite state automata and model checking, a well-known abstraction formalism and a formal verification framework, have provided a significant research spectrum for studying a variety of problems in computer science and modeling a wide range of real-world applications. Model checking and automata theory have evolved from traditional and theoretical paradigms to powerful and practical tools which have found applications in the industry. For example, Uppaal[19], ForSpec[21], and Spin[12]. Furthermore, timed automata have particularly emerged as a practical tool for verifying programs' correctness and as a light-weight version of formal methods for software development. A major direction of research on timed automata has extensively targeted real-time computational systems, and real-time model checking (see e.g.,[3,5,11,14,18,20]). In recent years, much research has been devoted to the dual connection between model checking and automata theory, leading to significant new results and major research directions in software engineering. Alternation is as a natural generalization of nondeterminism and a powerful concept for formalizing parallelism. This lead us to define alternating finite automata (AFA)[2,4,7] which have been extended with clock variables, in almost the same way that timed automata[9] extend nondeterministic finite automata. Alternation and other forms of alternat-

* Corresponding author. Tel.: +0-506-648-2302 ; fax: +0-506-648-2302.
E-mail address: fellah@unb.ca

1877-0509 © 2015 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Peer-review under responsibility of organizing committee of The 2015 International Conference on Soft Computing and Software Engineering (SCSE 2015)
doi:10.1016/j.procs.2015.08.409

ing finite automata (see for e.g., [2,14,17]) have extensively been investigated in formal languages and particularly used to alleviate certain problems and anomalies, potentially to the state-space explosion problem, that may rise in software systems. Unlike timed automata, parallel model of computations such iterative array and tree automata seem to be the less developed even though there has been recently a growing interests in mutual comparisons of different real-time state machines [6,10,13]. Iterative array automata that we refer to as iterative arrays (IA) are one-dimensional one-way infinite array of identical finite-state machines (*i.e.,* cells) that operate synchronously at discrete time steps. Other structures such as a n-dimensional iterative arrays (n-IA) and iterative tree automata also referred to as iterative trees (IT) have been studied and formalized in research. IT are systolic tree automata in which the cells are connected in an infinite full binary tree structure and each cell communicates with its parent and children. Generalizations, limitations, and language recognition capabilities of these models have been investigated in research, see for e.g., [10,13].

Propositional temporal logic [8,16,20] is one of the most widely used design and well-known specification language for modeling the properties of reactive and real-time systems. Methods for specifying and verifying programs have been formalized in terms of temporal logic properties, geared toward a fully algorithmic approach of low computational complexity as compared to program verification, formal methods, and theorem proving. Linear time temporal logic (LTL) and computational tree logic (CTL) formulas are the most common approaches used as verification tools and have been studied extensively in the literature [11,15,17,18]. One of the most prominent real-time specification formalism is metric temporal logic (MTL), a real-time extension of LTL in which time-constrained versions of some temporal operators has been expressed. Two commonly semantic domains are adopted in MTL, namely, pointwise semantics and continuous semantics. The main drawback is the logic in continuous semantics is undecidable though a restriction on the semantics. This had lead to versions of MTL which have been shown to be decidable [9,20]. In addition, MTL in pointwise semantics have been shown to be decidable [2]. Whereas, continuous semantics are more expressive than pointwise semantics in specifying the behavior of real-time systems where time constraints on events and system response time are considered to assert a formula property. Model checking evolves around some important automata-theoretical concepts - nondeterminism, alternation and timing on finite or infinite words, and propositional temporal formulas are often translated into different variety of automata [14,16,17,20]. In simple terms, the model checking can be stated as follows: "given a finite structure $\mathcal{S}$ and a formula φ in a propositional temporal logic, is $\mathcal{S}$ a model of φ"?

In this paper we first investigate a class of synchronous parallel models of real-time computation, iterative arrays and trees, then we study some of their capabilities and limitations in recognizing languages and solving real-time applications. In addition, we consider the dual connection between real-time models of computation and metric temporal logic. In particular, we show that both timed event alternating automata and timed metric temporal logic support the specifications of real-time systems altogether with a set of properties through two powerful dimensions – alternation and time. In a class by itself, such powerful machines do not only describe a theoretical model framework for formal languages but establishes strong foundations and relationships to timed propositional temporal logic. The rest of the paper is organized as follows. Section 2 presents real-time iterative arrays and trees, describes their characteristics, and proves their relationships. Relationships of a new determinizable subclass of alternating timed automata and specifications of formal translation of MTL, the most common approaches used in software verification, are detailed in Sections 3 and 4. Finally, a conclusion is provided in Section 5.

2. Real-time Iterative Arrays and Trees

Let Σ be a finite alphabet. We denote by $\mathbb{R}^+$ and $\mathbb{N}$ the set of all non-negative reals and positive natural numbers, respectively. The empty word is denoted by λ and $|w|$ indicates the length of the underlying word w. Thus, $|\lambda| = 0$. We denote by Σ^* the set of all words over Σ, where the operation $*$ refers to the Kleene closure operation on formal languages. Given an alphabet Σ, a language over Σ is a subset of Σ^*. An iterative array (IA) is a one-dimensional semi-infinite array (infinite on one end) of identical finite-state machines (*i.e.,* cells) that operate synchronously at discrete time steps. Each cell communicates with its left and right neighbors, except that the left cell, called special cell, communicates with the external world and its neighbors. At the beginning (time 0), each cell is in a distinguished quiescent state q_λ. During the computation, each cell of the IA communicates with its neighbors. The next state of a cell depends on the current state of a cell, the current state of the left neighbor (or external input for the special cell) and the current state of the right neighbor (see Figure 1 (a)).

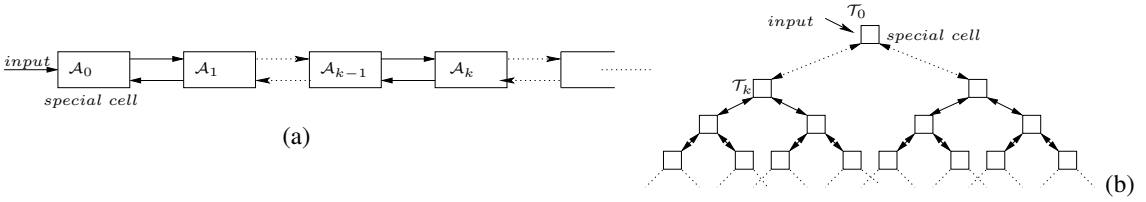

Fig. 1. (a) An Iterative Array (IA) $\mathcal{A}$; (b) An Iterative Tree (IT) $\mathcal{T}$.

Let $\mathcal{A}$ be an iterative array. An input word is of the form $w = a_1a_2\ldots a_n\$$ is fed to $\mathcal{A}_0$, $a_i \in \Sigma$, $n \geq 0$. The language L accepted by $\mathcal{A}$ is the set of all accepted words. Let w be a word such that $|w| = n$, an iterative array $\mathcal{A}$ is $T(n)$-time bounded (or has space complexity $T(n)$) if it accepts (or rejects) inputs of length n in more than $T(n)$ time steps. Similarly, we say that $\mathcal{A}$ is $S(n)$-space bounded (or has space complexity $S(n)$) if it accepts (or rejects) inputs of length n in more than $S(n)$ cells. Let $T : \mathbb{N} \to \mathbb{N}$ such that $T(n) \geq n+1$. If $T(n) = n+1$, then $\mathcal{A}$ is said to operate in real-time and called a real-time iterative array. Furthermore, languages accepted by $\mathcal{A}$ are called real-time languages, denoted by $L_{rt}(\mathcal{A})$. If $T(n) = cn + d$, for some constant c and d, L are linear-time languages denoted by $L_{lt}(\mathcal{A})$.

Definition 2.1. *An iterative array (IA) is a quintuple $\mathcal{A} = (Q, \Sigma, \delta_0, \delta, F)$, where (i) Q is a finite set of states; (ii) Σ is the input alphabet; (iii) $\delta_0 : (\Sigma \cup \{\$\} \times (Q \cup \{q_\lambda\})^2 \to Q \cup \{q_\lambda\}$ is the transition function of the special cell; (iv) $\delta : (Q \cup \{q_\lambda\})^3 \to Q \cup \{q_\lambda\}$ is the state transition function for every other cell except the special cell, which satisfies $\delta(q_\lambda, q_\lambda, q_\lambda) = q_\lambda$; (v) $F \subseteq Q$ is the set of final states; $q_\lambda \notin Q$ is a special quiescent state; $\$ \notin \Sigma$ is the input end-marker.*

An iterative tree (IT) is a systolic system in which the cells are connected in an infinite full binary tree structure and each cell communicates with its parent and children. Let $\mathcal{T}$ be an iterative tree, the root of IT is the special cell. The next state of the root is determined by the current state of the root, the input symbol, and the current states of its children. The input word is serially fed to the root (see Figure 1 (b)). In a similar mannecaptionAn Iterative Tree (IT) $\mathcal{T}$. r to iterative arrays we can define the notion of acceptance, rejection, and time (space) complexity of iterative trees. The language accepted by the real-time iterative tree $\mathcal{T}$ is referred to as the real-time language, denoted by $L_{rt}(\mathcal{T})$. Similarly, linear-time languages of IT can be defined. $\mathcal{T}$ has depth complexity $D(n)$ if for any word, w, that is accepted (or rejected) $\mathcal{T}$ uses at most $S(n)$ levels of the tree.

Definition 2.2. *An iterative tree (IT) is a six-tuple $\mathcal{T} = (Q, \Sigma, \delta_0, \delta_l, \delta_r, F)$, where (i) Q is a finite set of states; (ii) Σ is the input alphabet; (iii) $\delta_0 : (Q \cup \{q_\lambda\}) \times (\Sigma \cup \{\$\}) \times (Q \cup \{q_\lambda\})^2 \to Q \cup \{q_\lambda\}$ is the transition function of the root cell; (iv) $\delta_l, \delta_r : (Q \cup \{q_\lambda\})^4 \to Q \cup \{q_\lambda\}$ is the state transition functions of the left and right cells, respectively; (v) $F \subseteq Q$ is the set of final states. The symbol $\$ \notin \Sigma$ is the input end-marker. $q_\lambda \notin Q$ is a special quiescent state. $\delta_l(q_\lambda, q_\lambda, q_\lambda, q_\lambda) = \delta_r(q_\lambda, q_\lambda, q_\lambda, q_\lambda) = q_\lambda$.*

Theorem 2.1. *$T(n)$-time iterative array languages are accepted by $T(n)$-time iterative tree.*

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta_0, \delta, F)$ be an IA. We define a configuration $\widehat{C_t}$ of $\mathcal{A}$ as a snapshot computation of $\mathcal{A}$ at some instant in time, $t \geq 0$. Such a computation is described by a global state, and specified by a global transition which is induced by δ_0 and δ. We denote such global state and transition at time $t \geq 0$ by $\mathbb{G}_t$ and Δ, respectively.

Let $\mathbb{G}_t$ denote the global state at time $t \geq 0$ from $\mathcal{A}_i \to Q$ which labels each $\mathcal{A}_i$ by a set of states from Q. The initial global state, $\mathbb{G}_0$, includes all the cells $\mathcal{A}_i$, $i \geq 0$ which are in the quiescent state. The accepting global state is the root cell in F. Define $\Delta(\mathbb{G}_t, a) = \mathbb{G}_{t-1}$, where $\mathbb{G}_t$ and $\mathbb{G}_{t-1}$ are two global states and $a \in \Sigma$ expressed through the transition functions, δ_0, δ_l, and δ_r, any possible neighborhood into a corresponding global state. A configuration $\widehat{C_t}$ is described by the tuple $(\mathbb{G}_t, w)$, where $w \in \Sigma^*$ and $t \geq 0$. The initial configuration at time 0, $\widehat{C_0} = (\mathbb{G}_0, w)$, for all cells $\mathcal{A}_i$, $i \geq 0$. Borrowing the notation of[6], we investigate the transition from the global mapping of the iterative array $\mathcal{A}$ at time t to the mapping of $\mathcal{A}$ at time $t+1$, we are interested in the mapping Δ: $\mathbb{G}_t \to \mathbb{G}_{t-1}$ (*i.e.,* Δ: $\mathbb{G}_{t+1} \to \mathbb{G}_t$). For notational convenience, we use the former notation. Define $(\mathbb{G}_t, a) = \mathbb{G}_{t-1}$. For any $w \in \Sigma^*$ such that $|w| = l$, the recursive global transition Δ is defined as follows:

$$\Delta^l(\mathbb{G}_t, w) = \begin{cases} \mathbb{G}_0 & \text{if } l = 0 \\ \Delta(\Delta^{l-1}(\mathbb{G}_{t-1}, w'), a) & \text{if } l = |w|,\ w = w'a \\ \Delta(\Delta^{l-1}(\mathbb{G}_{t-1}, w), \$) & \text{if } l > |w| \end{cases}$$

where $l \in \mathbb{N}$, $w' \in \Sigma^*$, and $\Delta(q_0q_1 \dots q_n, a) = \delta_0(q_0, q_1, a)\ \ \delta(q_0, q_1, q_2) \dots \delta(q_{n-1}, q_n, q_\lambda)\ \ \delta(q_n, q_\lambda, q_\lambda)$. Furthermore, at time t and for $k \geq 1$ configurations are given as follows:

$$\widehat{C}_t(\mathcal{A}_0) = \delta_0(\{\Sigma \cup \$\}, \widehat{C}_{t-1}(\mathcal{A}_0), \widehat{C}_{t-1}(\mathcal{A}_1)), \dots, \widehat{C}_t(\mathcal{A}_k) = \delta(\widehat{C}_{t-1}(\mathcal{A}_{k-1}), \widehat{C}_{t-1}(\mathcal{A}_k), \widehat{C}_{t-1}(\mathcal{A}_{k+1}))$$

In order to simulate a real-time iterative array on a real-time iterative tree, we consider a restricted version of the regular IT denoted by $\mathcal{T}'$. We only traverse the leftmost path (or the rightmost path) of the tree $\mathcal{T}'$ during the computations. The two transition functions of the restricted $\mathcal{T}'$ are defined as $\delta_0 : \Sigma \times Q \times Q \rightarrow Q$ which is the transition function of the root cell. In δ_0, we consider the current states of its current input symbol, the root cell, and its left child. The second transition function is defined as $\delta_l : Q^3 \rightarrow Q$ which is the transition function of the left cell. In δ_l, we consider the current states of the cell, its parent, and its left child. The accepting global state is the root cell in F. Similarly to the iterative array $\mathcal{A}$ and by considering only the leftmost path of $\mathcal{T}'$, we can define $\Delta^l(\mathbb{G}_t, w)$ and $\widehat{C}_t(\mathcal{T}_k)$ where $k \geq 1$ such that:

$$\widehat{C}_t(\mathcal{T}_0) = \delta_0(\{\Sigma \cup \$\}, \widehat{C}_{t-1}(\mathcal{T}_0)), \widehat{C}_{t-1}(leftchild(\mathcal{T}_1))), \dots, \widehat{C}_t(\mathcal{T}_k) = \delta_l(\widehat{C}_{t-1}(\mathcal{T}_k), \widehat{C}_{t-1}(parent(\mathcal{T}_k)), \widehat{C}_{t-1}(leftchild(\mathcal{T}_k)))$$

□

From the above proof, the following results are stated.

Corollary 2.1. *Any $S(n)$-space iterative array language is accepted by $\log S(n)$-space iterative tree.*

Definition 2.3. *A function f is said to be IT-time-constructible if and only if there exists an iterative tree $\mathcal{T} = (Q, \Sigma, \delta_0, \delta_l, \delta_r, F)$ such that for all time steps $t \in \mathbb{N}$, $\widehat{C}_t(\mathcal{T}_0) \in F \iff \exists n \in \mathbb{N}$ such that $t = f(n)$)*

Corollary 2.2. *Let $\mathcal{A} = (Q, \Sigma, \delta_0, \delta, F)$ and $\mathcal{T} = (Q, \Sigma, \delta_0, \delta_l, \delta_r, F)$ be an iterative array and tree, respectively. Then the class of languages $L_{rt}(\mathcal{A})$ is properly included in the class of languages $L_{rt}(\mathcal{T})$.*

Proof. We need to prove that there exists a language that is accepted by a real-time IT but not by a real-time IA. For convenience we consider a version of the same language of[6,13] to prove that the proper containment of $L_{rt}(\mathcal{A})$ in $L_{rt}(\mathcal{T})$. Let such a language be $L_{rt} = \{\$u_k\$ \dots \$u_1\#v_1\$ \dots \$v_k\$$ such that $1 \leq k,\ u_i = v_iw_i,\ u_i,\ v_i,\ w_i \in \{a, b\}^*,\ 1 \leq i \leq k\}$. $L_{rt} \in L_{rt}(\mathcal{T})$, but $L_{rt} \notin L_{rt}(\mathcal{A})$. □

Furthermore, real-time IT languages are closed under Boolean operations.

Theorem 2.2. *The class of languages $L_{rt}(\mathcal{T})$ is closed under Boolean operations.*

Proof. We show that real-time iterative trees are closed under boolean operations such as union, intersection, concatenation, and Kleene star operations. Let $L_{rt}(\mathcal{T}_1)$ and $L_{rt}(\mathcal{T}_2)$ be two real-time languages accepted by $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. We can show that, $L_{rt}(\mathcal{T}_1) \cup L_{rt}(\mathcal{T}_2)$ are real-time languages. We construct a new IT, $\mathcal{T} = (Q, \Sigma, \delta_0, \delta_l, \delta_r, F)$, where every cell is augmented with two communicating channels, ch_1 and ch_2. Every run of $\mathcal{T}$ is executed along the channels which follow the computational events of $\mathcal{T}$, and where ch_1 and ch_2 simulate $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. The computational behaviors of $\mathcal{T}_1$ and $\mathcal{T}_2$ consist of their local states as well as the contents of the channels. Then, the final computational results are merged at the root of $\mathcal{T}$ using the union operation. We apply a similar argument for the intersection operation.

Now, we show by construction the concatenation operation, that is we construct an iterative tree $\mathcal{T}$ that accepts the language $L_{rt}(\mathcal{T}_1)$ concatenated with $L_{rt}(\mathcal{T}_2)$. Let $w = uv$ a string which can be decomposed into randomly two generated substrings u and v, where $|u| = l_1$ and $|v| = l_2$, respectively. The first $|u|$ inputs are fed from the root to the left subtrees and the remaining $l_2 =$ are fed to the right subtrees. The process of the decomposition and processing inputs is done recursively and produces the tree required by the automaton $\mathcal{T}$. Thus, all left children at every level of the tree simulate $\mathcal{T}_1$ on the l_1 symbol inputs, and all right children simulate $\mathcal{T}_1$ on the l_2 symbol inputs. Thus, $\mathcal{T}$ accepts the input if and only one there exists one constructed sub-iterative tree that accepts, but every other sub-iterative tree IT must reject for the overall $\mathcal{T}$ to reject. For the Kleene closure operation, let $L_{rt}(\mathcal{T})$ be a real-time language accepted by $\mathcal{T}$. We construct a new ITA $\mathcal{T}'$ such that such that $L_{rt}(\mathcal{T}) = L^*_{rt}(\mathcal{T}')$. The method of concatenating two real-time languages can be naturally extended to prove the Kleene star iteration. It is well known that the class of timed regular languages is not closed under complementation. □

3. Timed Event Alternating Automata

Timed alternating finite automata (TAA) have been independently introduced in[4,7] and throughly investigated in the literature. A timed event over a finite alphabet Σ is a pair $\rho = (\sigma, \tau)$, where $\sigma = \sigma_1\sigma_2 \dots \sigma_n$ is a non-empty finite event over Σ and τ is a time sequence such that $|\sigma| = |\tau|$. Thus, a finite timed event over Σ is an finite sequence $\rho = (\sigma_1, \tau_1)(\sigma_2, \tau_2) \dots (\sigma_n, \tau_n)$ of symbols σ_i paired with non-negative numbers τ_i. The expression $\tau_{i+1} - \tau_i$ indicates the time delay between two successive events $i + 1$ and i such that $1 \leq i \leq |\rho|$. Let $\mathbb{T} \subseteq \mathbb{R}^+$, we denote by $\Sigma^*_{\mathbb{T}}$ the set of all finite timed events over Σ. Such a set of timed events forms a timed language denoted by L. In the same way, infinite timed events can be defined similarly over Σ.

Let $\mathcal{X}$ be a set of finite clock variables (or clocks for short), the set $\Phi(\mathcal{X})$ of clock constraints ϕ over $\mathcal{X}$ is defined by the grammar: $\phi := x \bowtie c \mid \phi_1 \wedge \phi_2 \mid true \mid false$, where $c \in \mathcal{X}$, $c \in \mathbb{N}$ such that $c \geq 0$, $\bowtie \in \{<, \leq, =, >, \geq\}$, and $\wedge$ stands for the *and* logical operator. We denote by the symbol $\mathbb{B}$ the Boolean semiring, $\mathbb{B} = (\{0, 1\}, \wedge, \vee, ^-)$, where $\wedge$, $\vee$, and $^-$ denote the conjunction, disjunction and negation, respectively. Let Q be a set. Then $\mathbb{B}^Q$ is the set of all mappings of Q into $\mathbb{B}$; note that $u \in \mathbb{B}^Q$ can also considered as a Q-vector over $\mathbb{B}$. $(\mathbb{R}^+)^X$ indicates a vector with $|\mathcal{X}|$ elements (all non-negative) which refers to all real functions from $\mathcal{X}$ to $\mathbb{R}^+$. Now, we define timed alternating automata.

Definition 3.1. *A timed alternating finite automaton (TAA) is a six-tuple $\mathcal{M} = (Q, q_0, \Sigma, \mathcal{X}, \delta, F)$ where Q is a finite set of states and $q_0 \in Q$; Σ is a finite input alphabet; $\mathcal{X}$ is a finite set of clock variables; and $\delta : (Q \times \Sigma \times \Phi(\mathcal{X})) \mapsto \mathbb{B}^Q(Q \times \mathbb{P}(\mathcal{X}))$ is a finite partial function and where $\mathbb{P}$ denotes the power set; and $F \subseteq Q$ is the set of final states.*

Corollary 3.1. [4] *Timed alternating automata languages are closed under Boolean operations.*

The underlying structures of these parallel computation models could also be array-connected or tree-connected networks of different types of timed alternating automata with serial or parallel input. We refer to iterative arrays of timed alternating automata as IA_{TAA}, and iterative trees of timed alternating automata as IT_{TAA}. Based on the above discussion, Theorem 2.1, Corollary 2.1, and the fact that timed event transition systems are semantically equivalent to timed automata[14], we state the following claims:

Theorem 3.1. *IA_{TAA} $S(n)$-space are simulated by IT_{TAA} in $T(n)$-time and $\log(S(n))$-depth.*

Corollary 3.2. *The class of languages definable by IA_{TAA} and IT_{TAA} are affectively closed under Boolean operations.*

Due to space constraints, we omit the proofs of these claims.

Definition 3.2. *A timed event alternating automaton (TEAA) is a seven-tuple $\mathcal{E} = (Q, \Sigma, S, \mathcal{X}, h, g, F)$, where Q is a finite set, the set of states; Σ is an alphabet, the input alphabet; $S \subseteq Q$ is the set of starting states; $\mathcal{X}$ is a finite set, the set of clocks; h is a time transition function of $(\mathbb{B}^Q \times (\mathbb{R}^+)^X) \times \mathbb{R}^+$ into $(\mathbb{B}^Q \times (\mathbb{R}^+)^X)$; g is an event transition function from Q into the set of all mapping of $(\mathbb{B}^Q \times (\mathbb{R}^+)^X) \times \Sigma$ into $(\mathbb{B}^Q \times (\mathbb{R}^+)^X)$; $F \subseteq Q$ is the set of final states.*

$S = \{s_q \mid q \in Q, s_q = 1\}$. That is, for each s_q, $s_q = 1$ if and only if $q \in S$. In this paper, we assume that we only have one single starting state. That is, the number of s_q which are set to 1 is exactly equals to 1. Therefore, timed event alternating automata that we consider in this paper are deterministic.

We now turn to defining the sequential behavior of a TEAA. More specifically, the function h is defined over a timed event $\rho = (\sigma, \tau) = (a, t)$ and can be rewritten as:

$$h((q_1, q_2, \dots, q_{|Q|}, x_1, x_2, \dots, x_{|X|}), (a, t)) = ((q_1, q_2, \dots, q_{|Q|}, x_1 + t, x_2 + t, \dots, x_{|X|} + t), a)$$

where $q_i \in Q$ for $1 \leq i \leq |Q|$, $x_j \in \mathcal{X}$ for $1 \leq j \leq |\mathcal{X}|$, $a \in \Sigma$, and $t \in \mathbb{R}^+$ such that $t \geq 0$. Furthermore and without loss of generality, we can extend h to the set of all timed events, the set of all mappings.

Now define $f \in \mathbb{B}^Q$ by the condition $f_q = 1 \iff q \in F$, where f is called the *characteristic vector* of F. Define $s \in \mathbb{B}^Q$ by the condition $s_q = 1 \iff q \in S$, where s is called the *characteristic vector* of S. Let $\mathbb{v}_0$ be the *initial* characteristic vector of all clock evaluations. Define $\widehat{s} = s \cup \mathbb{v}_0$.

Definition 3.3. *Let $\mathcal{E} = (Q, \Sigma, S, g, h, \mathcal{X}, F)$ be a timed event alternating automaton and $w \in (\Sigma \times \mathbb{R}^+)^*$ be a timed event. w is accepted by $\mathcal{E}$ if and only if $f(g(h(\widehat{s}, w))) = 1$, where f is the characteristic vector of F, $\widehat{s} = s \cup \mathbb{v}_0$, s is the characteristic vector of S, $\mathbb{v}_0$ is the initial characteristic vector of all clock evaluations* (at the start, $\mathbb{v}_0 = 0$). Moreover, for each s_q, $q \in Q$, $s_q = 1$ if and only if $q \in S$; and for each s_x, $s_x = 0$, where $x \in \mathcal{X}$.

4. From MTL to Timed Event Alternating Automata

Linear-time propositional temporal logic (LTL)[1,3,16,18] is an extension of propositional logic that has originally targeted the correctness and prove of concurrent programs. LTL formulas are composed of atomic propositions, Boolean logic operators, and future time (dual past time) operators. That is, LTL components are made of atomic propositions, Boolean operators ($\vee$, $\wedge$, $\neg$), unary temporal operators X (*Next*); F (*eventually*), for example F_α, states that "α will be true (sometime in the strict future)"; G (*always*), for example G_α, states "α will always hold (in the strict future)"; and the binary infix temporal operator U (*until*), for example "I will eat until I become full". Several other metric temporal logics and different types of automata have been developed in research.

Metric temporal logic (MTL) was proposed in[15,20] as a step stone for the verification of real-time systems. MTL extends LTL by supporting the specification of relative time and real time constraints. Let $\mathsf{AP} = p_1, \ldots, p_n$ be a nonempty set of atomic propositions, where $p \in \mathsf{AP}$. There are a variety of metric temporal logics in the literature and with two interpretations of the semantics, pointwise or continuous. Most of the work in the last two decades has been based on the pointwise (*i.e.,* discrete) semantics and the specification algorithms have been rooted in discrete temporal techniques such as automata. Consequently, the formulas of MTL are built over Σ and interpreted over timed events or built over AP and interpreted over timed states. Unfortunately, it is undecidable to determine the satisfiability of a metric temporal constraint and model checking problems over state event semantics.Moreover, MTL are undecidable over timed event semantics if past time operators are included[9,18]. Since the main drawback in these semantics are the undecidability, we can restrict the continuous semantics by assuming that there is a finite number of discontinuities on a bounded interval I. We define the abstract syntax of metric temporal logic (MTL) over the alphabet Σ as follows:

$$\varphi := a \mid \top \mid \varphi_1 \wedge \varphi_2 \mid \neg\, \varphi \mid \varphi_1\, \mathsf{U}_I\, \varphi_2 \mid \mathsf{X}_I\, \varphi$$

where $a \in \Sigma$, and the operators $\top$ ($\bot$) indicate the true (false) propositions, respectively. $I \subseteq \mathbb{R}^+$ is an interval of the form $[l,u], [l,u), (l,u)$, or $(l,u]$, $0 \leq l \leq u$. The right-end bounded intervals may possibly be the infinity, ∞. The real-time metric temporal logic (MTL) formulas can be defined over AP using the Boolean operators ($\vee$,$\wedge$, $\neg$), time-constrained unary temporal operator (X_I (*Next*)), and time-constrained binary temporal operator (U_I (*Until*)).

In addition, other temporal operators can also be derived from standard conventions to enable the description of time-dependent events, in particular G and F. For instance, the constrained eventually operator $\mathsf{F}_I\varphi \equiv \top\mathsf{U}_I\varphi$, and the constrained always operators $\mathsf{G}_I\varphi \equiv \neg\mathsf{F}_I\neg\varphi$ where F_I and G_I are the time-constrained versions of the operators F and G, respectively. For example, $\mathsf{F}_{\leq c}\ p_1$ means the property p_1 holds at some future time within the next c time units, and $p_1\ \mathsf{U}_{\leq 5}\ p_2$ means p holds until some future time, at most 5 units in future, where p_2 must hold. We define a $\mathsf{dual}(\varphi)$ a formula obtained from φ by switching $\top$ and $\bot$, $\vee$ and $\wedge$, and by complementing subformulas of φ. Thus, denote $\mathsf{dual}(\mathsf{U}_I)$ by $\widetilde{\mathsf{U}_I}$, and $\varphi_1\widetilde{\mathsf{U}_I}\varphi_2 \equiv \neg(\neg\varphi_1\mathsf{U}_I\neg\varphi_2)$. In the same way, $\mathsf{dual}(\mathsf{X}_I)$ is denoted by $\widetilde{\mathsf{X}_I}$, and $\widetilde{\mathsf{X}_I} \equiv \neg\mathsf{X}_I\neg\varphi$. The model checking can simply be stated as follows: "Given a finite structure M and a formula φ in a propositional temporal logic, is M a model of φ"? We write $\langle M, q\rangle \models \varphi$ to indicate that the formula φ is satisfied at state q in the structure M. In[14] the relationship between timed state-event LTL and timed Büchi state-event automaton have been studied and established through languages over infinite words. Timed state-event LTL are defined over the set of atomic proposition AP and the alphabet Σ. Several results have been proved to show a corresponding between timed Büchi automata and model checking[3,14]. One such a result is reflected in the following:

Theorem 4.1. [14] *Let $A_M = (Q, Q_0, \mathsf{AP}, \delta_s, \Sigma, \mathcal{X}, \delta_e)$ be a timed labeled transition system describing the system M and φ a timed state-event LTL. Then there exists a timed Büchi state-event automaton $A_\varphi = (Q, Q_0, \mathsf{AP}, \delta_s, \Sigma, \mathcal{X}, \delta_e, F)$ such that $L(A_M) \subseteq L(A_\varphi)$, where $\Sigma = 2^{\mathsf{AP}}$ and $|Q| \leq 2^{O(|\varphi|)}$. Moreover, M satisfies φ if and only if $L(A_M) \cap L(A_{\neg\varphi}) = \emptyset$.*

Let $\rho = (\sigma, \tau) \in \Sigma^*_{\mathbb{T}}$ denotes an finite timed event over Σ, φ an MTL formula, $i \in \mathbb{N}$, $i \leq |\rho|$, Without loss of generality, the satisfaction relation $(\rho, i) \models \varphi$ (read "ρ at position i satisfies φ") is defined inductively as follows:

$(\rho, i) \models a$ iff $\sigma_i = a$
$(\rho, i) \models \top$
$(\rho, i) \models \varphi_1 \wedge \varphi_2$ iff$(\rho, i) \models \varphi_1)$ and $(\rho, i) \models \varphi_2)$
$(\rho, i) \models \neg\varphi)$ iff$\rho, i \not\models \varphi$
$(\rho, i) \models \mathsf{X}_I\ \varphi)$ iff$(\rho, i+1) \models \varphi, \tau_{i+1} - \tau_i \in I$

$$(\rho, i) \models \varphi_1 \mathsf{U}_I \varphi_2 \quad \text{iff} \quad \exists\, j \geq i \mid (\rho, j) \models \varphi_2,\ \tau_j - \tau_i \in I, \text{ and } (\rho, k) \models \varphi_1, \forall\, k \mid i \leq k < j.$$

The language defined by an MTL formula φ in pointwise semantic is given by $L(\varphi) = \{\, \rho \in \Sigma^*_{\mathbb{T}} \ : \ (\rho, 0) \models \varphi \,\}$. Furthermore, we assume the other satisfaction relations dual until ($\widetilde{\mathsf{U}_I}$), dual next ($\widetilde{\mathsf{X}_I}$), constants and connectives (*i.e.,* $\vee, \top, \ldots$) could be defined in a natural way.

Definition 4.1. *Let ρ be a timed event and φ an MTL formula. ρ satisfies the MTL formula φ, written as $\rho \models \varphi$, if and only if $(\rho, 0) \models \varphi$. Assuming that the first event is an initial event that occurs at time 0.*

Now, we consider a formula φ then we construct a TEAA, $\mathcal{E}_\varphi = (Q_\varphi, \Sigma_\varphi, S_\varphi, \mathcal{X}_\varphi, h_\varphi, g_\varphi, F_\varphi)$, such that a finite event is a model for φ if and only if it is accepted by $\mathcal{E}_\varphi$. That is, $L(\mathcal{E}_\varphi) = L(\varphi)$. Each state of $\mathcal{E}_\varphi$ is labeled with a propositional subformula. We replace the connectives in each metric temporal logic by a timed event alternating automaton, and then include these automata into a larger timed event alternating automaton that consider and cover the entire structure of the formulas. All MTL formulas are assumed to be in negation normal form (NNF). That is, all negations are pushed inwards and appear only on propositions. For example the following is a set of equivalences: $\neg\neg\varphi = \varphi$, $\neg\widetilde{\mathsf{X}_I}\varphi = \widetilde{\mathsf{X}_I}\neg\varphi$, $\neg(\varphi_1 \wedge \varphi_2) = (\neg\varphi_1) \vee (\neg\varphi_2)$, $\neg(\varphi_1 \vee \varphi_2) = (\neg\varphi_1) \wedge (\neg\varphi_2)$, $\neg(\varphi_1 \mathsf{U}_I \varphi_2) = (\neg\varphi_1)\ \widetilde{\mathsf{U}_I}\ (\neg\varphi_2)$, $\neg(\varphi_1\ \widetilde{\mathsf{U}_I}\ \varphi_2) = (\neg\varphi_1)\ \mathsf{U}_I(\neg\ \varphi_2), \cdots$.

We define the closure of φ, $cl(\varphi)$, as the smallest set of formula containing $\top$, all subformula of φ including the initial formula, any subformula $\psi_1 \mathsf{U}_I \psi_2$ or $\psi_1 \widetilde{\mathsf{U}_I} \psi_2$ of φ, any subformula of type $\mathsf{X}_I\varphi$ or $\widetilde{\mathsf{X}_I}\psi$ of φ. The closure $cl(\varphi)$ of an MTL formula φ describes and forms the minimal set of states of $\mathcal{E}_\varphi$. Let $initialformula(\varphi)$ denote the initial given formula φ. Then, Q_φ contains a location q_ψ for every subformula ψ of φ with $initialformula(\varphi)$ being the initial state of $\mathcal{E}_\varphi$ denoted by q_φ. That is, $Q_\varphi = \{q_\psi \ : \ \psi \in cl(\varphi)\} \cup q_\varphi$. Furthermore, $cl(\varphi)$ captures the following clauses:

(i)	$\varphi \in cl(\varphi)$	(ii)	if $\varphi_1 \in cl(\varphi)$ then NNF of $\varphi_1 \in cl(\varphi)$
(iii)	if $\varphi_1 \wedge \varphi_2 \in cl(\varphi)$ then $\varphi_1, \varphi_2 \in cl(\varphi)$	(iv)	if $\varphi_1 \vee \varphi_2 \in cl(\varphi)$ then $\varphi_1, \varphi_2 \in cl(\varphi)$
(v)	if $\psi = \varphi_1, \ldots, \varphi_n$ and $\varphi \in cl(\varphi)$ then $\psi \in cl(\varphi)$	(vi)	for each subformula ψ of $\varphi_1 \mathsf{U}_I \varphi_2$, $\psi \in cl(\varphi)$
(vii)	for each subformula ψ of $\mathsf{X}_I\varphi_2$, $\psi \in cl(\varphi)$		

Following [16], define a recursive function $\delta(q_\psi, a)$, for each subformula $\psi \in \varphi$ and $a \in \Sigma$ as follows:

$$\begin{aligned}
\delta(q_\varphi, a) &= x.\delta(\varphi, a)\\
\delta(q_{\psi_1} \vee q_{\psi_2}, a) &= \delta(q_{\psi_1}, a) \vee \delta(q_{\psi_2}, a)\\
\delta(q_{\psi_1} \wedge q_{\psi_2}, a) &= \delta(q_{\psi_1}, a) \wedge \delta(q_{\psi_2}, a)\\
\delta(q_{\psi_1 \mathsf{U}_I \psi_2}, a) &= ((x.\delta(q_{\psi_2}, a)) \wedge x \in I) \vee\\
&= ((x.\delta(q_{\psi_2}, a)) \wedge (q_{\psi_1 \mathsf{U}_I \psi_2}))\\
\delta(q_{\psi_1 \widetilde{\mathsf{U}_I} \psi_2}, a) &= ((x.\delta(q_{\psi_2}, a)) \vee x \notin I) \wedge\\
&= ((x.\delta(q_{\psi_2}, a)) \vee (q_{\psi_1 \widetilde{\mathsf{U}_I} \psi_2}))\\
\delta(q_{\mathsf{X}_I\psi}, a) &= x.\delta(q_\psi, a) \wedge (x \in I)\\
\delta(q_{\widetilde{\mathsf{X}_I}\psi}, a) &= x.\delta(q_\psi, a) \wedge (x \notin I)
\end{aligned}$$

Note that whenever a subformula ψ is fulfilled the automaton starts and resets the clocks. The set F_φ consists of all locations $q_{\psi_1 \mathsf{U}_I \psi_2} \in Q_\varphi$ that corresponds to the *until* subformulas of φ. It is easy to check that the resulting automaton $\mathcal{E}_\varphi$ is a TEAA. For any locations q_{ψ_1} and q_{ψ_2}, the definition $\delta(q_{\psi_1}, a)$ holds only if ψ_2 is a subformula of ψ_1. That is, $q_{\psi_1} \rightarrow q_{\psi_2}$. Moreover, $\Sigma_\varphi = 2^{\mathsf{AP}}$, $S_\varphi = q_\varphi$, $\mathcal{X}_\varphi = \mathcal{X}$, g_φ is semantically described above by δ and h_φ is simply defined as the function h_φ over Q_φ and the set of clocks $\mathcal{X}_\varphi$. The transition functions h_φ and g_φ are recursively defined over all subformulas, not just on $cl(\varphi)$.

Theorem 4.2. *Let $\mathcal{E}_\varphi = (Q_\varphi, \Sigma_\varphi, S_\varphi, \mathcal{X}_\varphi, h_\varphi, g_\varphi, F_\varphi)$ be a timed event alternating automaton and φ be a given MTL formula in negation normal form, then $L(\mathcal{E}_\varphi) = L(\varphi)$. Moreover, the size of $\mathcal{E}_\varphi$ is linear to the size of φ.*

Proof. First, we can show that $L(\mathcal{E}_\varphi) \subseteq L(\varphi)$. Let ρ be a timed event in $L(\mathcal{E}_\varphi)$ such that $|\rho| = n$ and assume $\mathcal{E}\varphi$ accepts ρ. We prove by induction on the structure of the formula that all subformulas $\psi \in cl(\varphi)$ and each $1 \leq i \leq n$, $(\rho, i) \models \varphi$. That is, $L(\mathcal{E}_\varphi) = L(\varphi)$. The base case can easily be verified. That is, either $\psi = a$ or $\psi = \neg a$ for the formula $a \in \Sigma$. The other cases that we need to prove by induction are when the subformulas $\psi = \mathsf{X}_I\varphi$ and $\psi = \varphi_1 \mathsf{U}_I \varphi_2$. By

induction hypothesis, it is simple to check that $(\rho, i+1) \models \varphi$, then $(\rho, i) \models \mathsf{X}_I\varphi$. The second case of the connective U_I is quite straightforward. The proof of the second inclusion, $L(\varphi) \subseteq L(\mathcal{E}_\varphi)$, considers the complement of $\mathcal{E}_\varphi$, that is, the deterministic timed event alternating automaton representing $\neg\varphi$ is the complement of the automaton representing φ, and moreover the set of states represented by $cl(\varphi)$ of $\overline{\mathcal{E}}_\varphi$ and $\mathcal{E}_\varphi$ are the same. This completes the proof. □

5. Conclusion

Time dependent models have become increasingly important in formal modeling and verification of software systems. We presented a class of theoretical time dependent models, compared their relative expressiveness as far as formal language recognition, and presented several relationship results across these models through corollaries and theorems. Even though iterative array automata, iterative tree automata, and alternating Turing machines share many common features and there exists mutual translations between them; they have not been developed independently to a large extent like timed automata. As a consequence of this, determinization, decidable, and undecidable problems related to these models are an interesting avenue for future work. Furthermore, the focus of this work considered MTL over finite timed events, introduced another interesting sub-class of determinizable timed alternating automata, and developed a construction for translating timed metric temporal logic to timed event alternating automata. An extension of this work is to consider MTL with past operators and rewrite the algorithm at the occurrence of past events. This will lead to whether the extension would increase the complexity of the algorithm compared with existing approaches.

References

1. F. Blahoudek, M. Kretinsky, J. Strejcek. Comparison of LTL to deterministic Rabin automata translators. *In Proc. of LPAR, Lecture Notes in Computer Science*(LNCS), Springer-Verlag; 2013, vol. 8312, 164–172. , .
2. M. Jenkins, J. Ouaknine, A. Rabinovich, J. Worrell. Alternating timed automata over bounded time. *In Proc. of Lecture in Computer Science* (LICS), 2010, 60–69
3. T. Babiak *et al*. LTL to Büchi Automata: translator: fact and more deterministic. *Lecture Notes in Computer Science,* (TACAS), 2012, vol. 7214, 95–109.
4. A. Fellah, C. Harding. Language equations for timed alternating finite automata, *Intern. J. Comput. Math.,* 2003, **80**(9), 1075–1091.
5. T. Babiak, F. Blahoudek, M. Kretinsky, J. Strejcek. Effective translation of LTL to deterministic Rabin automata: beyond the (F,G)-fragment, *In Proc. of ATVA* , Springer-Verlag; (LNCS), 2013, vol. 8172, 24–39.
6. K. Culik II, S. Yu. Iterative tree automata, *Theoretical Computer Science* 1986, vol. 32, 227–247
7. S. Lasota, I. Walukiewicz. Alternating timed automata, *ACM Trans. Comput. Logic*, 2008, **9**(2),
8. A. Pnueli, Y. Sa'ar, D.L. Zuck. Jtlv: A framework for developing verification algorithms, *In Proc. Int. Conf. on Comput. Aided Verification*, (CAV), 2010, 171–174.
9. R. Alur D. Dill. A theory of timed automata, *Theoreterical Computer Science,* 1994, **126**(2), 183–235.
10. C. Iwamoto, K. Tateishi, K. Morita, K. Imai. A quadratic speedup theorem for iterative arrays, *Acta Informatica*, 2009, **38**(11-12), 847–858.
11. A.E. Ben Salem, A. Duret-lutz, F. Kordon. Model checking using generalized testing automata, *Transactions on Petri Nets and Other Models of Concurrency (ToPNoC), VI,* Springer Verlag; 2012. (LNCS), vol. 7400, 94–122,
12. G.J. Holzmann. The SPIN model checker: prime and reference manual. Addison-Wesley Pearson Education, 2004.
13. M. Kutrib, A. Malcher. Real time reversible iterative arrays, *Theorettical Computer Science,* 2010, **411**(4-5), 812–822,
14. A. Fellah, Time and alternation: An automata based Framework to software model checking, *Proc. ACM Symp. Appl. Comput.*, (SAC), 2010, 2498–2502.
15. O. Maler and D. Nickovic, A. Pnueli. From MTL to timed automata *Proc. Int. Conf. Formal Modeling and Analys. Timed Syst.,* (FORMATS), (LNCS), Springer-Verlag; 2006, 4202, 274–289.
16. M. Hammer, A. Knapp, A. S. Merz. Truly On-The-Fly LTL model checking, *Intern. Conf. Tools and Algorithms for the Construction and Analysis of Systems*, Springer-Verlag; 2005, 191–205.
17. R. Pelanek and J. Strajcek, Deeper connections between LTL and alternating automata, *In Proc. Int. Conf. on Implementation and Application of Automata* (CIAA), (LNCS), 2006, 3845, 238–249.
18. Z. Weijun *et al*. Translating linear temporal logic formulas into automata, *Information Theory and Coding*, 2012, **9**(6), 100–113.
19. http://www.uppaal.com
20. D. Nickovic, N. Piterman. From MTL to deterministic timed automata, *In Proc. Int. Conf. Formal Modeling and Analys. Timed Syst.* (FORMATS), (LNCS), 2010, 152–167.
21. R. Armoni, L. Fix, A. Flaisher, *et al.*, The ForSpec temporal logic: A new temporal property-specification language *In Proc. Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, Springer-Verlag; 2001, **2280**(4), 211–296,